CS477/ECE478 - Formal Software Development Methods
– Spring 2007 –

Unit Project

April 15, 2007

1 Alternating Bit Protocol

from Wikipedia

Alternating bit protocol (ABP) means a simple data link layer network protocol that retransmits lost or corrupted messages.

Messages are sent from transmitter A to receiver B. Assume that the channel from A to B is initialized and that there are no messages in transit. Each message from A to B contains a data part and a one-bit sequence number, i.e., a value that is 0 or 1. B has two acknowledge characters that it can send to A: ACK0 and ACK1. We assume that the channel may corrupt a message and that there is a way in which A and B can decide whether or not they have received a correct message. How and to which extent that is possible is the subject of coding theory.

When A sends a message, it sends it continuously, with the same sequence number, until it receives an acknowledgment from B that contains the same sequence number. When that happens, A complements (flips) the sequence number and starts transmitting the next message.

When B receives a message that is not corrupted and has sequence number 0, it starts sending ACK0 and keeps doing so until it receives a valid message with number 1. Then it starts sending ACK1, etc.

This means that A may still receive ACK0 when it is already transmitting messages with sequence number one. (And vice-versa.) It treats such messages as negative-acknowledge characters (NAKs). The simplest behaviour is to ignore them all and continue transmitting.

The protocol may be initialized by sending bogus messages and acks with sequence number 1. The first message with sequence number 0 is a real message.

Specific assumptions. For the purpose of this project, let us assume that data part of a message sent contains an integer representing the id of the message, which is incremented with each new distinct message being send.

Furthermore, let us assume that the channel does not corrupt data; however, it might randomly loose data.

2 Requirements

1. (optional, but highly recommended) Specify the protocol above as a rewrite theory, specify the correctness requirement as an LTL formula and use Maude model checker to verify it.

2. To model the possibility that the channel randomly looses the messages being send, add to JavaFAN a "builtin" function `random()`, which non-deterministically returns either boolean *true* or *false*.

3. Implement the above protocol in Java using the above defined `random()`

4. and use JavaFAN to prove that the implementation provides sequenced data transfer from the A to B for a length of the message up to $N \in \overline{2,5}$. That is, B will not receive messages out of order; more precisely, whenever B receives a message with id n, it must already have received the message with id $n-1$.

5. Formally show, using JavaFAN, that in the above implementation it is virtually possible that not all messages send by A reach B. In particular, show that it is possible that B receives no message.

6. Modify the channel implementation such that it won't loose more than 3 messages in a row. Use JavaFAN to formally show that this implementation also guarantees the entire sequence of messages being send by A *is received* (in order) by B.

To allow an easy testing of the project, your JavaFAN program should be parameterized by a constant N of sort *Nat*, specifying the length of a message.